

Het genereren van willekeur

Over Tetris en lavalampen

Randomness, oftewel willekeur, wordt al jaren veelvuldig gebruikt door software. De meeste gebruikers en ontwikkelaars realiseren zich echter niet hoe moeilijk het is om willekeur te genereren.

Door Martijn van Steenbergem

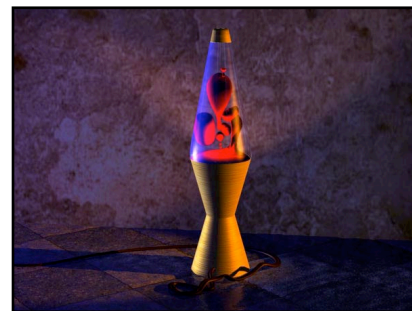
DAT KLINKT BEST wel tegenstrijdig, het 'genereren van willekeur'. Toch kan menig computerprogramma niet zonder. Denk maar aan je favoriete muziekspeler; als deze de liedjes door elkaar kan spelen ('shuffle'), heeft het willekeur nodig. Ook vele computerspelletjes zijn afhankelijk van randomness. Een welbekend voorbeeld is Tetris: in welke volgorde komen de verschillende blokjes naar beneden vallen? Als dit niet willekeurig gekozen werd, zou elk Tetris-spelletje dezelfde serie blokjes gebruiken en gaat een groot deel van het plezier van het spel verloren. Willekeurige getallen spelen ook een grote rol in de cryptografie; in dit vakgebied worden er zeer strenge eisen gesteld aan de gegenereerde getallen.

De statistiek is een ander vakgebied waar random getallen veel gebruikt worden, met name bij het simuleren van kansexperimenten. Volgens Wikipedia werd in 1927 de eerste random number table geconstrueerd: een grote tabel vol met willekeurige getallen. Het gebruik van zo'n tabel was meestal makkelijker dan bijvoorbeeld herhaaldelijk met een dobbelsteen gooien. Nu nog worden soms tabellen geleverd met de wiskundeboeken voor de middelbare school. Echter, naarmate de computer populairder werd, kwam men op het idee software te gebruiken om die lange lijsten met willekeurige getallen te maken. De eerste softwarematige *random number generator* (RNG) was de *middle-square method*, verzonden door de van oorsprong Hongaarse John von Neumann in 1946. Deze werkt als volgt: je begint met een eerste getal, zeg 7483. Vervolgens neem je daar het kwadraat van en pak je de middelste vier cijfers om het tweede getal te krijgen: 9952. Deze stap herhaal je net zoveel als je nodig hebt.

Er is echter een aantal problemen met deze methode. Ten eerste is de serie getallen die je krijgt *periodiek*: het herhaalt zichzelf om de zoveel tijd. Als de getallen uit maar vier cijfers bestaan, is de periode maximaal 10.000. Immers, een getal uit de serie heeft altijd precies dezelfde opvolger. In veel gevallen echter zal de periode veel kleiner zijn dan 10.000, vanwege de wiskundige aard van het algoritme. En als je eenmaal 0 (oftewel 0000) als uitkomst hebt gekregen, zullen alle getallen daarna ook 0 zijn. Een tweede nadeel is dat je kunt voorspellen wat het volgende getal wordt. Nou hoeft dit niet per se een probleem te zijn; als je Tetris speelt merk je toch niks van welke getallen er precies gebruikt worden, maar toch wil je over het algemeen dat je, gegeven een getal uit de serie, niet zomaar de volgende kunt voorspellen.

Als de middle-square method geen goede generator is, welke eigenschappen moet een RNG dan hebben om wel goed te zijn? De twee belangrijkste eigenschappen zijn onvoorspelbaarheid en uniformiteit. Uniformiteit wil zeggen dat de kans op alle mogelijke uitkomsten even groot is. In het geval van een digitale generator moet de kans op een 1 en een 0 dus allebei $1/2$ zijn.

In de loop der jaren zijn er vele andere algoritmes verzonden die erg dicht in de buurt komen van de bovengenoemde eigenschappen. Een algoritme kan echter nooit helemaal onvoorspelbaar zijn, juist omdat het een deterministisch programma is. Omdat het geheugen van een computer eindig is, is het aantal toestanden waarin een computer zich kan bevinden ook eindig. Computers zijn volledig deterministisch: als je de huidige toestand weet, weet je ook de volgende toestand. Daarom is een algoritmische RNG *altijd* periodiek. Deze soort generators wordt dan ook wel *pseudo random number generators* (PRNG) genoemd: ze kunnen echte randomness benaderen, maar het nooit helemaal bereiken. Natuurlijk kan de periode zo enorm groot zijn dat de kans dat je aan het einde van het unieke deel van de serie komt erg klein is. Voor



© Steve Turnbull

de meeste toepassingen zijn PRNGs dan ook voldoende.

Echte wiskundigen nemen met benaderingen echter geen genoegen. Daarom is de zoektocht naar *true* RNGs verder gegaan. Aangezien alleen software nooit tot ware randomness zal leiden, moet de oplossing buiten de computer gezocht worden. In 1996 kwam het bedrijf Silicon Graphics op het idee digitale foto's van lavalampen te gebruiken als bron van randomness. Omdat we met de huidige beschikbare apparatuur niet kunnen voorspellen hoe een chaotisch systeem zoals een lavalamp zich gedraagt, is hiermee aan de eis van onvoorspelbaarheid voldaan. Echter, omdat we het determinisme en de controle kwijt zijn, kunnen we geen enkele garantie meer geven over de data die deze bron levert, en dus ook niet over de uniformiteit van de data. Daarom wordt voordat de data omgezet wordt in getallen vaak een proces genaamd *deskewing* toegepast. Via vaak ingewikkelde wiskundige formules kan zo de kans op een 0 of 1 vrijwel of helemaal naar $1/2$ gebracht worden. In de praktijk vormen lavalampen en andere chaotische bronnen zeer robuuste RNGs.

Toch wordt meestal voor PRNGs gekozen, omdat deze vaak handzamer zijn: je hebt geen digitale camera of andere meetapparatuur nodig. Bovendien zijn ze voor hun snelheid alleen afhankelijk van de rekenkracht van de computer, en zijn ze voldoende om schijnbaar willekeurig Tetrisblokjes te kiezen of liedjes door elkaar te husselen. Alleen voor kritische applicaties zoals cryptografie zijn *true* RNGs rendabel.